

Test coverage para Laravel

verificar si xdebug esta instalado

```
php -v
```

Instalar xdebug

<https://xdebug.org/download>

renombrar el archivo en

```
C:\xampp\php\ext\php_xdebug.dll
```

en el php.ini añadir estas líneas

```
[xdebug]
zend_extension = xdebug
xdebug.remote_autostart = 1
xdebug.profiler_append = 0
xdebug.profiler_enable = 0
xdebug.profiler_enable_trigger = 0
xdebug.profiler_output_dir = "c:\xampp\tmp";
xdebug.profiler_output_name = "cachegrind.out.%t-%s"
xdebug.remote_enable = 1
xdebug.remote_handler = "dbgp"
xdebug.remote_host = "127.0.0.1"
xdebug.remote_log="c:\xampp\tmp\xdebug.txt"
xdebug.remote_port = 9000
xdebug.trace_output_dir = "c:\xampp\tmp";
xdebug.remote_cookie_expire_time = 36000
xdebug.mode=coverage
xdebug.start_with_request=yes
```

Generar Reporte de Cobertura

Para generar un informe de cobertura de código, usa PHPUnit con el indicador

`--coverage`.

Generar un informe en consola:

```
Copiar código  
php artisan test --coverage
```

Resultado

```
Console\Kernel ..... 16 / 66.7%  
Exceptions\Handler ..... 100.0%  
Http\Controllers\Api\JournalController ..... 100.0%  
Http\Controllers\Controller ..... 100.0%  
Http\Controllers\JournalController ..... 100.0%  
Http\Kernel ..... 100.0%  
Http\Middleware\Authenticate ..... 0.0%  
Http\Middleware\EncryptCookies ..... 100.0%  
Http\Middleware\PreventRequestsDuringMaintenance ..... 100.0%  
Http\Middleware\RedirectIfAuthenticated ..... 0.0%  
Http\Middleware\TrimStrings ..... 100.0%  
Http\Middleware\TrustHosts ..... 0.0%  
Http\Middleware\TrustProxies ..... 100.0%  
Http\Middleware\ValidateSignature ..... 100.0%  
Http\Middleware\VerifyCsrfToken ..... 100.0%  
Models\Journal ..... 100.0%  
Models\User ..... 100.0%  
Providers\AppServiceProvider ..... 100.0%  
Providers\AuthServiceProvider ..... 100.0%  
Providers\BroadcastServiceProvider ..... 0.0%  
Providers\EventServiceProvider ..... 100.0%  
Providers\RouteServiceProvider ..... 100.0%  
View/Components\Footer ..... 100.0%  
  
Total: 76.9 %
```

Generar un informe en HTML:

```
Copiar código  
php artisan test --coverage-html=coverage-report
```

Los resultados estarán disponibles en la carpeta `coverage-report`.

Resultado

| | Code Coverage | | | | | | | | |
|--------------|------------------------|---------|---------|------------------------|---------|---------|------------------------|---------|--------|
| | Lines | | | Functions and Methods | | | Classes and Traits | | |
| Total | <div><div></div></div> | 76.92% | 40 / 52 | <div><div></div></div> | 75.00% | 15 / 20 | <div><div></div></div> | 61.54% | 8 / 13 |
| ▀ Console | <div><div></div></div> | 66.67% | 2 / 3 | <div><div></div></div> | 50.00% | 1 / 2 | <div><div></div></div> | 0.00% | 0 / 1 |
| ▀ Exceptions | <div><div></div></div> | 100.00% | 2 / 2 | <div><div></div></div> | 100.00% | 1 / 1 | <div><div></div></div> | 100.00% | 1 / 1 |
| ▀ Http | <div><div></div></div> | 67.86% | 19 / 28 | <div><div></div></div> | 62.50% | 5 / 8 | <div><div></div></div> | 40.00% | 2 / 5 |
| ▀ Models | | n/a | 0 / 0 | | n/a | 0 / 0 | | n/a | 0 / 0 |
| ▀ Providers | <div><div></div></div> | 88.24% | 15 / 17 | <div><div></div></div> | 85.71% | 6 / 7 | <div><div></div></div> | 80.00% | 4 / 5 |
| ▀ View | <div><div></div></div> | 100.00% | 2 / 2 | <div><div></div></div> | 100.00% | 2 / 2 | <div><div></div></div> | 100.00% | 1 / 1 |

Legend

Low: 0% to 50% **Medium:** 50% to 90% **High:** 90% to 100%

Generated by [php-code-coverage 10.1.16](#) using [PHP 8.1.12](#) and [PHPUnit 10.5.38](#) at Mon Nov 18 17:02:42 UTC 2024.

Midiendo el type coverage con PEST en Laravel



Raúl López
<https://raullg.com>

En primer lugar, tenemos que instalar Pest y su extensión para el type coverage:

Para instalar Pest:

```
composer require pestphp/pest --dev --with-all-dependencies
```

```
./vendor/bin/pest --init
```

Para instalar el plugin:

```
composer require pestphp/pest-plugin-type-coverage --dev
```

Una vez instalado todo, podemos medir el type coverage de nuestro código sin escribir ni un solo test a través del siguiente comando:

```
./vendor/bin/pest --type-coverage
```

Esto nos dará un output tal que así:

```
app/SourceCode/BitbucketProvider.php ..... 100%
app/SourceCode/Contracts/AccountInfoProvider.php ..... 100%
app/SourceCode/Contracts/DownloadsZipFile.php ..... 100%
app/SourceCode/Contracts/ReceivesZipFile.php ..... 100%
app/SourceCode/Contracts/SourceCodeProvider.php ..... 100%
app/SourceCode/Contracts/HandlesWebhook.php ..... 100%
app/SourceCode/Contracts/RegistersWebhook.php ..... 100%
app/SourceCode/S3ZipProvider.php ..... 100%
app/SourceCode/GitLabProvider.php ..... 100%
app/SourceCode/LocalFolderProvider.php ..... 100%
app/SourceCode/GitHubProvider.php ..... 100%
app/Console/Commands/CreateDemoRepositories.php ..... 100%
app/Console/Commands/GenerateProjectDocumentation.php ..... 100%
app/Console/Kernel.php ..... 100%
Total: 100.0 %
```

Una captura del type coverage de CodexAtlas

Además, configurando esto como un paso en nuestro pipeline, podemos **requerir** un mínimo para nuestro type coverage, y así asegurarnos de que cualquier actualización de código tenga que estar tipado al máximo posible:

```
./vendor/bin/pest --type-coverage --min=70
```

En caso de no pasar el type coverage, obtendríamos algo así:

```
ERROR Type coverage below expected: 94.6%. Minimum: 100.0%
```

Esto, al ejecutarse, devolverá un código de respuesta distinto de cero, que será interpretado por las líneas de comando como un error y no permitirá pasar el pipeline.